Performance Evaluation of FMOSSIM, a Concurrent Switch-Level Fault Simulator*

Randal E. Bryant
Carnegie-Mellon University
Dept. of Computer Science
Pittsburgh, PA 15213

Michael D. Schuster
California Institute of Technology
Computer Science 256-80
Pasadena, CA 91125

Abstract

This paper presents measurements obtained while performing fault simulations of MOS circuits modeled at the switch level. In this model the transistor structure of the circuit is represented explicitly as a network of charge storage nodes connected by bidirectional transistor switches. Since the logic model of the simulator closely matches the actual structure of MOS circuits, such faults as stuck-open and closed transistors as well as short and open-circuited wires can be simulated. By using concurrent simulation techniques, we obtain a performance level comparable to fault simulators using logic gate models. Our measurements indicate that fault simulation times grow as the product of the circuit size and number of patterns, assuming the number of faults to be simulated is proportional to the circuit size. However, fault simulation times depend strongly on the rate at which the test patterns detect the faults.

1. Introduction

Concurrent simulation techniques [1] have long been used to enhance the performance of gate-level fault simulators. Such programs simulate the good circuit in its entirety and keep track of how the behavior of each faulty circuit differs from that of the good circuit by selectively simulating portions of the faulty circuit. It appears to the user as if many circuits are being simulated at once, but the computational effort is greatly reduced from that of *serial* simulation, in which each faulty circuit is simulated separately. Published performance figures for concurrent simulators [2] indicate that the fault simulation time for a large circuit with many faults remains within a factor of 8 of the simulation time for the fault-free circuit alone.

There has been a growing recognition within the testing community [3] that logic gate simulators do not adequately model the behavior of MOS circuits, especially in the presence of such nonclassical faults as stuck-open and closed transistors, and short and open-circuited wires. These faults can cause even a simple logic gate to become a seemingly complex sequential device. As an alternative, researchers have suggested modeling MOS circuits at a level of detail that more accurately represents their electrical characteristics [4].

We recently implemented a fault simulator for MOS digital circuits [5] based on the switch-level model, [6] with the transistor structure of the circuit represented explicitly, but with each transistor modeled in a highly idealized way. This approach has proved successful for logic simulation in programs such as MOSSIM II, [6] because such properties as the bidirectional nature of MOS transistors and the ability of nodes to store charge are modeled explicitly, rather than by some artificial translation into logic gates. Unlike the time-consuming algorithms used by circuit simulators, switch-level simulators model the circuit in a sufficiently simplified way that they operate at speeds comparable with conventional logic gate simulators. The switch-level logic model is well suited for modeling a variety of failures in MOS circuits in a reasonably realistic way, because many faults can be viewed as creating new switch-level networks which differ from the switch-level representation of the good circuit. Hence, while the switch-level model has proved successful for logic simulation, it seems especially attractive for fault simulation.

Our program FMOSSIM utilizes concurrent simulation techniques to minimize the amount of time spent simulating each faulty circuit. In contrast to other concurrent fault simulators for MOS, [7] our simulator places no restrictions on the topology of the transistor networks. Adapting the concurrent technique to such a switch-level simulator required major changes to the data structures and method of processing events. With such a program, many questions arise regarding its performance, since a fault simulator must achieve very high performance if it is to be applied to circuits of VLSI complexity. In particular, we would like to answer such questions as:

- How are fault simulation times affected by the circuit size, number of test patterns, and number of faults?
- How can the properties of the test patterns affect fault simulation times?
- How would fault simulation times be affected if we simulate only a random sample of the possible faults?

*This research was supported at Caltech by the IBM Corporation and by the Defense Advanced Research Projects Agency, ARPA Order 3771. Michael Schuster was supported in part by a Bell Laboratories Ph.D. Scholarship.

0738-100X/85/0715$01.00 © 1985 IEEE

We measured the performance of FMOSSIM while fault simulating circuits of moderate complexity (up to 1148 transistors.) We will present the details of this investigation in this paper. Our most important findings are summarized as follows:

- Fault simulation times grow as the product of the circuit size and the number of patterns, assuming the number of faults grows with the circuit size. This represents a significant improvement over the performance of a serial fault simulator, which requires time proportional to the product of the circuit size, number of patterns, *and* number of faults.

- Fault simulation times are considerably better for test patterns that detect the most severe faults quickly.

- Under random fault sampling, the simulation time grows linearly with the sample size.

2. Network Model

FMOSSIM implements the same network model as the logic simulator MOSSIM II [6]. This model and its characteristics have been discussed in detail elsewhere, and we will only describe it briefly here. A switch-level network consists of a set of *nodes* connected by a set of *transistors*. No restrictions are placed on how the nodes and the transistors are interconnected. Each node has a state 0, 1, or X, where 0 and 1 represent low and high voltages, respectively. The X state represents an indeterminate voltage arising from an uninitialized node, from a short circuit, or from improper charge sharing.

Each node is classified as either an *input* node or a *storage* node. An input node provides a strong signal to the network, as does a voltage source in an electrical circuit. Its state is not affected by the actions of the network. Examples include the power and ground nodes Vdd and Gnd, which act as constant sources of the states 1 and 0, respectively, as well as any clock or data inputs. The state of a storage node is determined by the operation of the network. A storage node holds its state when not connected to input nodes, much as a capacitor in an electrical network. To model the effects of charge sharing, each storage node is assigned a discrete *size* (from a small set of possible values), where a larger storage node is assumed to have much greater capacitance than a smaller one. Most circuits can be modeled with just two node sizes, with high capacitances nodes such as busses assigned larger size values than all other nodes.

A transistor is a device with terminals labeled *gate*, *source*, and *drain*. No distinction is made between the source and drain connections – each transistor is symmetric and bidirectional. Transistors can be either *n-type*, *p-type*, or *d-type* so that both nMOS and CMOS circuits can be modeled. A d-type transistor corresponds to a negative threshold depletion mode device. A transistor acts as a resistive switch connecting or disconnecting its source and drain nodes according to its type and the state of its gate node, as shown in Table 1. Transistor states 0 and 1 represent open (nonconducting) and closed (fully conducting) conditions, respectively. The X state represents an indeterminate condition between open and closed, inclusive.

gate state	n-type	p-type	d-type
0	0	1	1
1	1	0	1
X	X	X	1

Table 1 Transistor State as Function of Gate Node State

To model ratioed circuits, each transistor is assigned a discrete *strength* from a small set of values, where a stronger transistor is assumed to have much greater conductance than a weaker one. The total number of strengths required depends on the circuit to be modeled. Most CMOS circuits do not utilize ratioed logic and hence can be modeled with just one transistor strength. Most nMOS circuits require only two strengths, with pull-up loads assigned a weaker strength than all other transistors. However, we can introduce additional strengths to model more peculiar circuit structures or to model fault effects.

3. Fault Injection

The switch-level model can represent the behavior of a MOS circuit in the presence of a variety of node, transistor, and wire faults. FMOSSIM directly implements both node and transistor faults, where a node fault causes the node to behave as an input node set to the specified state, while a transistor fault causes the transistor to be permanently stuck-open or stuck-closed, without changing its strength. Other fault types can be injected by inserting extra *fault transistors* in the network, much like the method proposed by Lightner and Hachtel. [8] For example, a short circuit can be represented by a transistor of very high strength between the two nodes that is set to 1 in the faulty circuit and 0 in the good circuit. Similarly, an open circuit can be represented by splitting a node into two parts connected by a transistor of very high strength where this transistor is set to 1 in the good circuit and 0 in the faulty circuit. Most significantly, injecting these faults requires no modeling capabilities beyond those already possessed by the switch-level model.

4. Concurrent Simulation

Our switch-level algorithm computes the behavior of a circuit for each change in network inputs by repeatedly computing the *steady state response* [6] of the network until a stable state is reached. Each computation of the steady state response involves setting the transistors according to the states of their gate nodes and determining the new states that would appear on the storage nodes due to the connections to input nodes and to other storage nodes. To exploit the locality of activity in a logic circuit, only node states in the *vicinity* of a *perturbed* node are computed, where a node is said to be perturbed if it is the source or drain of a

transistor that has changed state, or if it is connected by a conducting (either 1 or X) transistor to an input node that has changed state. The vicinity of a node consists of the set of all storage nodes connected by paths of conducting transistors that do not pass through input nodes. This definition exploits the *dynamic* locality in the network where the source and drain of a transistor in the 0 state are considered to be electrically isolated. In contrast, earlier switch-level simulators [9] exploited only the *static* locality in the where the network was partitioned only according to its DC-connected components. Typically, a vicinity contains only a few nodes, and hence activity remains highly localized.

The scheduling of activities in a switch-level simulator proceeds much like that in an event-driven functional level simulator – the simulation of a logic element causes one or more nodes to change state, and the simulator schedules activities for those logic elements affected by these changing states. In a switch-level simulator, however, the "logic elements" are transistor vicinities, i.e. sets of nodes connected by transistors in the 1 or X state. The boundaries between these logic elements depend on the current state of the network and change during the course of the simulation. This property presents the major challenge to applying concurrent simulation techniques – the boundaries between the logic elements can be different in the different circuits being simulated.

In the conventional form of the concurrent algorithm, [1] a list is maintained for each logic element indicating the states of inputs in the fault-free circuit and in each faulty circuit for which the states differ from those of the fault-free circuit. The scheduling of events proceeds as with a conventional logic simulator, but the processing of an event involves computing the outputs of the element for every input combination on the list. Such an approach will not work with our switch-level fault simulator, because of the dynamic and data-dependent nature of the logic element boundaries. Instead, we maintain a separate state list for each node, containing records of the form $\langle i,s_i \rangle$, indicating that in circuit i (each circuit is represented by an integer ID with the good circuit having ID 0), this node has state s_i. Such records are maintained only for the good circuit, and for those circuits i such that $s_i \neq s_0$. Events are scheduled on a circuit-by-circuit basis. That is, an "event" specifies both a node and a circuit indicating that the state of this node must be recomputed in this particular circuit. To simulate a single time step the program first simulates all activities for the good circuit. These simulations can create additional events for the faulty circuits, because a node in a faulty circuit that previously had the same state as the good circuit may now be different. Following the good circuit simulation, the program simulates the activities for each faulty circuit in turn. By simulating each circuit separately, we can exploit the locality of activity in the individual circuits even though this locality is data-dependent. By keeping the state and event lists sorted according to the circuit ID's, and maintaining "shadow pointers" pointing to the current positions on the state lists, we can minimize the time spent searching these lists.

5. Performance Results

To evaluate the performance of FMOSSIM, we simulated two dynamic RAM circuits for different numbers of faults, and different test sequences. All measurements were taken on a VAX 11/780 running Berkeley 4.2 Unix for a version of the program written in C. The first circuit, RAM64, contains 378 transistors and 229 nodes, while the second, RAM256, contains 1148 transistors and 695 nodes. The circuits incorporate a variety of MOS structures such as logic gates, bidirectional pass transistors, dynamic latches, precharged busses, and three-transistor dynamic memory elements. Memory circuits were chosen because they could easily be scaled in size, and because they could be fully tested by test sequences consisting of special tests of the control and peripheral logic followed by a marching test [10] of the memory array. In terms of the performance of a switch-level simulator, these circuits provide rather difficult test cases, because the bit lines act as large global busses, and hence activity is not well localized. When faults such as stuck-at-one control lines occur, the locality is further reduced. Furthermore, while memory circuits exhibit a high degree of controllability, their observability is low, because there is only a single output.

The circuits were simulated for randomly chosen subsets of the following fault classes: single storage nodes stuck-at-zero, single storage nodes stuck-at-one, and single pairs of adjacent bit lines shorted together. To validate the program, we also simulated other faults, including stuck-open and stuck-closed transistors. The performance characteristics for such faults did not differ significantly from those of node faults.

Figure 1 illustrates the typical behavior of FMOSSIM when simulating a large number of faults. It shows the data for a simulation of RAM64 with 428 faults over a sequence of 407 patterns consisting of 7 patterns to test the control and peripheral logic, 40 patterns to perform a marching test of the row select logic, 40 patterns to perform a marching test of the column select and bit line logic, and 320 patterns to perform a marching test of the memory array. Each "pattern" here actually represents a sequence of 6 input settings to cycle the clocks. Any time the simulation of a faulty circuit produces a result on the output data pin different than the good circuit simulation, the fault is considered detected, and the simulation of that circuit is dropped.

The rising curve in Figure 1 indicates the cumulative number of faults detected as the simulation proceeds. The falling curve indicates the CPU time required to simulate each pattern. This curve divides into two parts: the "head" consists of the first 87 patterns during which all faults in the control and bus logic are detected, followed by the "tail" during which the faults in the memory array are detected. The simulation time starts at 45

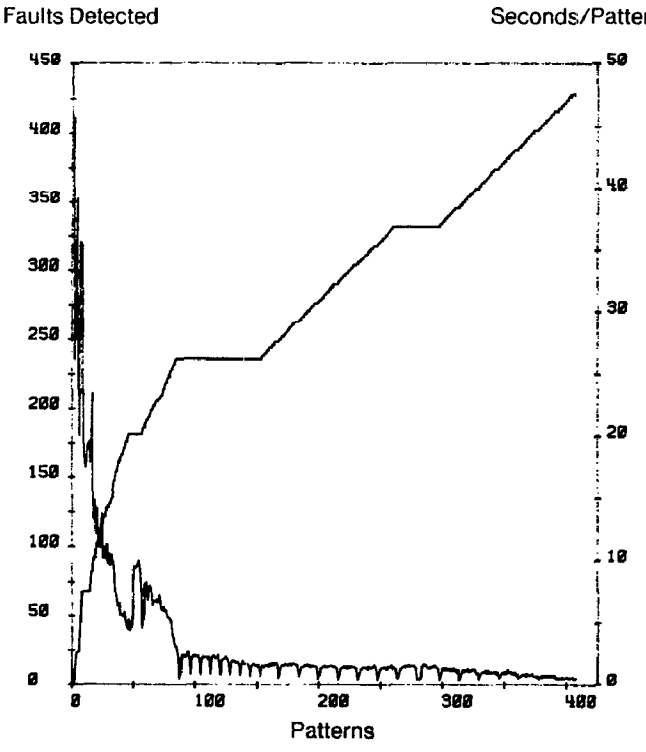

Fig. 1 Test Sequence 1 for RAM64

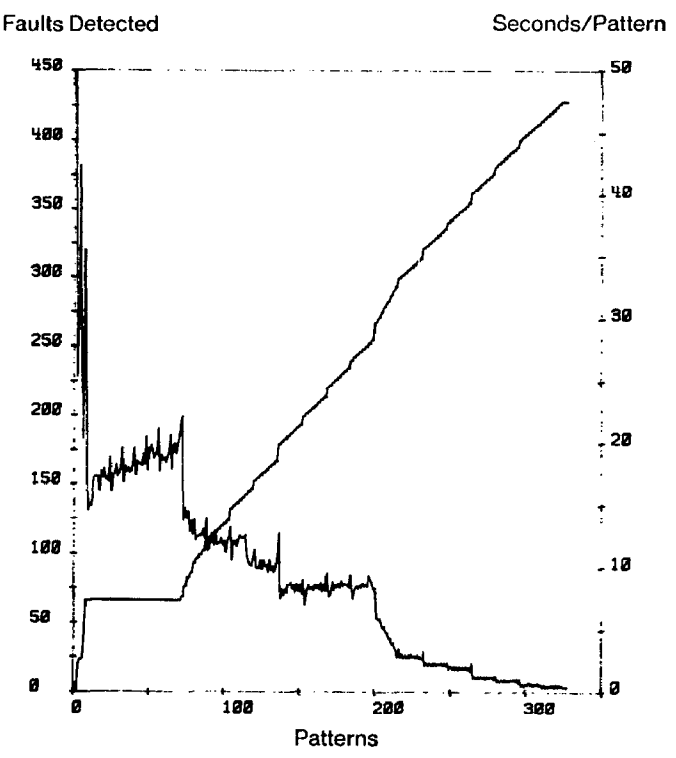

Fig. 2 Test Sequence 2 for RAM64

seconds per pattern while the circuit is initialized and major faults such as frozen clock lines are being simulated. Those faults that create behavior vastly different from that of the good circuit and hence require the most additional effort to simulate are detected quickly. Once these faults are dropped the performance improves markedly. During the tail portion the simulator runs on average just 3 times slower than it would to simulate only the good circuit, even though as many as 190 circuits are being simulated simultaneously. The faulty circuits remaining during this portion behave much like the good circuit, because they contain only bit errors in the memory, which have no effect unless the faulty bit is selected. The entire fault simulation requires 21.9 minutes of CPU time, with 71% of the time consumed during the first 87 patterns. In contrast, the simulation of the good circuit alone requires 2.7 minutes, and a serial fault simulation in which each faulty circuit is simulated individually until it produces an output different from that of the good machine would require 404 minutes (6.7 hours).** This performance ratio of 18 for concurrent versus serial simulation is gained largely during the tail end of the simulation, when many faults can be simulated concurrently at little additional cost.

Figure 2 illustrates how the choice of test sequence can affect the performance of the simulator. This simulation is the same as before, except that the patterns to test the row and column logic were omitted, leaving a total of 327 patterns. As a consequence, except for the 65 faults detected during the first seven patterns, all other faults are detected slowly as the marching test of the memory array proceeds, including faults in the address decoding and bus control logic. The time per pattern drops more slowly than before, because many faults that cause behavior much different from that of the good machine remain undetected for a long time. This simulation required 49 minutes of CPU time, even though the test sequence is shorter than before. Serial simulation, on the other hand, would require 448 minutes (7.5 hours), and hence concurrent simulation has a performance ratio of only 9, due largely to the lack of a tail end effect. This result shows that the shortest test sequence for a set of faults may not give the shortest simulation time, and that the penalty is worse for concurrent simulation than for serial. On the other hand, most test engineers look for a test sequence that detects many faults quickly, and this helps the concurrent simulator.

To see how the simulation time scales with the size of the circuit, we simulated RAM256 for a test sequence consisting of 1447 patterns similar to the first test sequence applied to RAM64. Simulating the good circuit alone requires 25.3 minutes for this sequence.

**All serial fault simulation times were estimated by summing over all faults the number of patterns required to detect the fault times the average time to simulate the good circuit for 1 pattern.

To run the test for all 1362 possible single stuck-at and single bus short faults, concurrent simulation requires a total of 202 minutes (3.4 hours), while serial simulation would require 15,169 minutes (10.4 days!) Comparing these results to the time required for RAM64, we see that both the time to simulate the good circuit alone and the time for concurrent simulation has scaled up by a factor of 9, while the time for serial simulation has scaled by a factor of 37. This result makes concurrent simulation seem increasingly attractive as circuits grow larger. It shows that concurrent simulation time scales as the size of the circuit times the number of patterns, assuming the number of faults is proportional to the circuit size. Serial simulation time, on the other hand, scales as the product of all three factors.

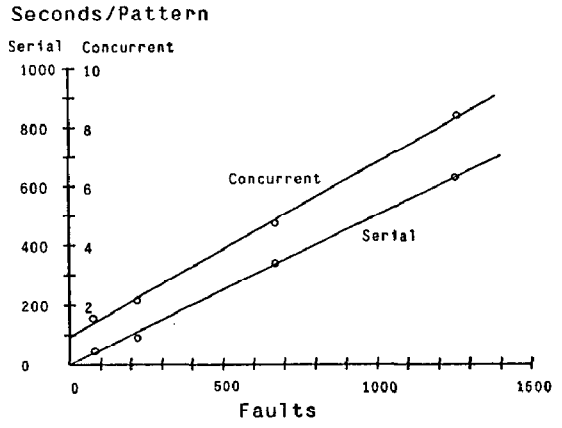

Fig. 3 Avg. Time per Pattern vs. Number of Faults for RAM256

Figure 3 shows the results of simulating RAM256 for different numbers of randomly selected faults, where times are measured in the average number of seconds per clock cycle over the entire length of the simulation. Note that the scale for serial simulation is 100 times that for concurrent. Both concurrent and serial simulation show a linear dependence on the number of faults, with serial being 85 times slower than concurrent. The linear growth of concurrent simulation can be viewed as both good and bad. On one hand, it shows that we pay no penalty for the overhead of maintaining the node states as lists that must be searched to find the states for a given circuit. On the other hand, it shows that our simulator exploits only the commonality between each faulty circuit and the good circuit. In many cases, two faulty circuits will behave more nearly like each other than like the good circuit, but we do not exploit this. Such a mechanism, if implemented without excessive overhead, could improve the performance even further.

6. Conclusion

Our experience with FMOSSIM has shown that it is a very useful tool for developing test sequences. Even when developing a test for a small section of an integrated circuit (such as an ALU or a register array), the fault simulator provides information that is hard to obtain by any other means. It quickly directs the designer to those areas of the circuit that require further tests. For example, in developing test sequences for the memory design described previously, we discovered that a simple marching test provided high coverage in the memory array itself, but that testing the control logic and peripheral circuits such as the input and output latches was more difficult.

As the size of a circuit increases, both good circuit and concurrent fault simulation times scale quadratically, because both the time per pattern and the number of patterns scale linearly. Serial simulation, on the other hand, scales cubically, because the number of faults also increases. FMOSSIM provides this performance while also providing a more accurate model of the circuits, especially when realistic faults are present.

References

1. E. Ulrich and T. Baker, "The Concurrent Simulation of Nearly Identical Digital Networks", *IEEE Computer*, April 1974, pp. 39-44.

2. E. Ulrich, *et al*, "High-Speed Concurrent Fault Simulation with Vectors and Scalars", *Twentieth Design Automation Conference*, July 1983.

3. R. Wadsack, "Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits", *Bell System Technical Journal*, Vol. 57, May-June 1978, pp. 1449-1473.

4. J. Hayes, "A Fault Simulation Methodology for VLSI", *Nineteenth Design Automation Conference*, July 1982, pp. 393-399.

5. M. Schuster, and R. Bryant, "Concurrent Fault Simulation of MOS Digital Circuits", *Advanced Research in VLSI*, P. Penfield,ed., MIT, January 1984, pp. 129-138.

6. R.E. Bryant, "A Switch-Level Model and Simulator for MOS Digital Systems", *IEEE Transactions on Computers*, Vol. C-33, No. 2, February 1984, pp. 160-177.

7. A. Bose, *et al*, "A Fault Simulator for MOS LSI Circuits", *Nineteent Design Automation Conference*, July 1982, pp. 400-409.

8. M. Lightner and G. Hachtel, "Implication Algorithms for MOS Switch Level Functional Macromodeling, Implication, and Testing", *Nineteenth Design Automation Conference*, July 1982, pp. 691-698.

9. R.E. Bryant, "MOSSIM: A Switch-Level Simulator for MOS LSI", *Eighteenth Design Automation Conference*, July 1981, pp. 786-790.

10. S. Winegarden and D. Pannell, "Paragons for Memory Test", *International Test Conference 1981*, IEEE, 1981, pp. 44-48.